GLOBAL COMMAND AND CONTROL SYSTEM (GCCS)
Mobile Code
Security Policy Guidance
For
Browser Script Software Development
(JavaScript, JScript, VBScript)

Prepared By:

National Security Agency
9800 Savage Road
Ft. Meade, MD 20755

30 April 1999

Approved by:

____
Jeff Watkins
Test Director

Approved by:

____
LtCol Mike Lopez
GCCS Security Officer

Form SF298 Citation Data

Report Date ("DD MON YYYY") 30041999	Report Type N/A	Dates Covered (from... to) ("DD MON YYYY")

Title and Subtitle Global Command and Control System (GCCS) Mobile Code Security Policy Guidance For Browser Script Software Development (JavaScript, JScript, VBScript)	**Contract or Grant Number**
	Program Element Number
Authors	**Project Number**
	Task Number
	Work Unit Number
Performing Organization Name(s) and Address(es) National Security Agency 9800 Savage Road Ft. Meade, MD 20755	**Performing Organization Number(s)**
Sponsoring/Monitoring Agency Name(s) and Address(es)	**Monitoring Agency Acronym**
	Monitoring Agency Report Number(s)
Distribution/Availability Statement Approved for public release, distribution unlimited	
Supplementary Notes	
Abstract	
Subject Terms "IATAC COLLECTION"	
Document Classification unclassified	**Classification of SF298** unclassified
Classification of Abstract unclassified	**Limitation of Abstract** unlimited
Number of Pages 30	

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 074-0188*

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	4/30/99	Report

4. TITLE AND SUBTITLE
Global Command & Control System: Mobile Code Security Policy Guidanc Fo

5. FUNDING NUMBERS

6. AUTHOR(S)
NSA

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

IATAC
Information Assurance Technology Analysis Center
3190 Fairview Park Drive
Falls Church VA 22042

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Defense Technical Information Center
DTIC-IA
8725 John J. Kingman Rd, Suite 944
Ft. Belvoir, VA 22060

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT	12b. DISTRIBUTION CODE
	A

13. ABSTRACT ***(Maximum 200 Words)***
Information contained in classified computer systems requires protection above and beyond that required by commercial users. Hence, Web access, and the browsers that implement that access, must be closely monitored and guarded against both intentional and unintentional disclosure, alteration, and/or destruction of information. Global Command and Control System (GCCS) users also need protection from disruption and denial of service. This document defines policies for the use of browser script languages, including JavaScript, JScript and VBScript, in web pages developed for use on the GCCS, and defines procedures for the application of those policies to determine the suitability of code for use within classified enclaves manipulating highly sensitive information.

14. SUBJECT TERMS	15. NUMBER OF PAGES
JAVA Script,Information Security,JScript,JBScript	
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	UNCLASSIFIED	UNCLASSIFIED	None

TABLE OF CONTENTS

1 INTRODUCTION

Browser scripting languages are extensions to the original hyper-text markup language (HTML) that are understood by the user's web browser. These extensions give web pages the power that today's web users require by integrating together browser plug-ins, ActiveX controls, Java applets, and user forms into an interactive web browsing experience. There are three languages commonly used for browser scripting: JavaScript, JScript, and VBScript. JavaScript is a cross-platform; object based scripting language created by Mr. Brendan Eich of Netscape Corporation for its Navigator client and server applications. JavaScript was first known as Mocha, then LiveScript, before the name JavaScript was finally selected in 1996. JScript and VBScript are Microsoft's implementation of browser scripting. JScript is an adaptation of the original JavaScript, while VBScript is an adaptation of Microsoft's popular Visual Basic programming language to web browser scripting. While the term JavaScript was chosen in 1996 to describe the new scripting language's similarity to the general-purpose language Java, this common name has often resulted in more confusion than clarity. The two languages are technically independent and serve different purposes, as described in Appendix B of this document. Whereas Java is a general-purpose programming language that can be easily used over a network, JavaScript – and its siblings JScript and VBScript – is an extension to the HTML that is used for publishing information on the World Wide Web (WWW). This scripting extension permits web pages incorporating browser scripts to provide the user additional functionality, and a richer user interface, than could be accomplished using HTML alone.

While the additional capabilities that browser scripting provides to programmers and users alike are advantageous, they also make it possible for unscrupulous programmers to access features of the users' web browser that perhaps should not be made available to them. Widespread publicity is given to these security holes by the technical press, resulting in a constant race between programmers and system administrators to protect their data against hackers equally intent on compromising their systems.

Information contained in classified computer systems requires protection above and beyond that required by commercial users. Hence, Web access, and the browsers that implement that access, must be closely monitored and guarded against both intentional and unintentional disclosure, alteration, and/or destruction of information. Global Command and Control System (GCCS) users also need protection from disruption and denial of service. This document defines policies for the use of browser script languages, including JavaScript, JScript and VBScript, in web pages developed for use on the GCCS, and defines procedures for the application of those policies to determine the suitability of code for use within classified enclaves manipulating highly sensitive information.

Throughout the remainder of this document, the terms *script* or *browser script* shall be used to refer collectively to Netscape JavaScript, Microsoft JScript, and Microsoft VBScript, except where the three are differentiated. When they must be referred to separately, they are referred to as *Netscape JavaScript* or *JavaScript*, *Microsoft JScript*, or *JScript*, and *Microsoft VBScript*, or *VBScript*.

1.1 BACKGROUND

Browser scripts are cross-platform; object based scripting extensions to the HTML that is used for defining pages for viewing on the WWW. Scripting languages are used as “cut-down programming” tools to execute simple, often repetitive tasks. It allows web pages to have dynamic functionality, including active controls, checking of on-line forms and the ability to upload and download files from remote web servers. This added functionality, although transparent to the user, requires considerable effort on the part of web developers, who must contend with multiple versions of scripting languages, incompatible browser features, and implementation inconsistencies.

Browser scripting works only on browsers that implement a scripting engine. There are a large number of versions of scripting engines that have been incorporated into the different versions of Internet Web browsers from all vendors. Most prominent of these vendors are Netscape and Microsoft. The first scripting engine to be commercially used was JavaScript version 1.0, released in 1996 and incorporated into the Netscape 2.0 browser. At that time, the Microsoft browser, Internet Explorer 2.0, did not support any type of scripting extensions. Not to be outdone, Microsoft Internet Explorer 3.0, released in 1997, supported a new scripting extension based upon Netscape’s version 1.0 JavaScript. Since Microsoft did not have a license to the JavaScript name, they named their scripting language JScript. At this time, Netscape had moved on to JavaScript 1.1, which was incorporated into their Navigator 3.0 browser. Microsoft’s early implementation of JScript was significantly less capable than Netscape’s offering, but Microsoft’s clout in the marketplace put it in a position to compete directly with Netscape for the control of the browser market, touching off the now infamous “Browser Wars”. Since the beginning of the browser wars, both companies have refined their technologies considerably. Microsoft’s Internet Explorer 4.0 and Netscape’s Navigator 4.0 both implement supersets of the Version 1.2 JavaScript language, with over 90% commonality between the two implementations. Additionally, Microsoft’s browser supports the VBScript scripting language, providing developers with more flexibility (and confusion). Hence, a carefully crafted web page that uses the language capabilities that are common to both browsers can have a consistent look and functionality, regardless of the browser that is used to view it.

Both Netscape and Microsoft have collaborated with the European Computer Manufacturer’s Association (ECMA) to standardize the JavaScript / JScript scripting languages. The result of this collaboration, ECMAScript, is functionally identical to the original JavaScript Version 1.1 language found in the Netscape Navigator 2.0 and Microsoft Internet Explorer 3.0 browsers. While the ECMAScript standard gives web developers a well-known standard that is assured to work on both Microsoft and Netscape browsers, current browser versions offer significant functionality increases over ECMAScript. Netscape’s Navigator 4.5, released in late 1998, implements version 1.3 of JavaScript, and Microsoft’s Internet Explorer 5.0, scheduled for release in early 1999, implements version 1.3 of JScript and version 5.0 of VBScript. Both Java-derived scripting languages are two generations beyond the ECMAScript 1.1 standard that is their common subset. Once again, though, web programmers can enjoy a commonality between their current implementations that is over 90%.

Regardless of the implementation, the scripting languages give the web programmer a great deal of power. Much of this power, such as the ability to highlight a button on the screen,

is benign. However, some features, such as the ability to turn off the browser's menu bar, or upload a file from the user's computer, can be easily misused to cause aggravation, denial of service, compromises, or corruption of data on a user's machine.

1.2 SCOPE

This document addresses security concerns associated with the use of the browser scripting languages within the GCCS environment (intentional malicious activity, unintentional user errors, or oversights in development) and establishes policies for browser script code that is incorporated into web pages that are viewed using GCCS Web browsers. Malicious activities could possibly be performed by individuals with access to Secret Internet Protocol Router Network (SIPRNET) or GCCS, outside hackers, or perhaps hackers employed by hostile organizations or countries. The risks associated with internally developed and deployed browser scripts can be mitigated through the proper application of this policy.

While the security policies put forward here can do little to stop the individual who desires access to GCCS, the proper application of these policies can remove a potential avenue of attack. The risks from internally developed GCCS scripting languages would also be limited. These policies are intended for the following audiences:

- GCCS proponent
- GCCS developer
- GCCS integrator/implementor
- GCCS certifier

It should also be noted that some commands and/or communities have banned certain scripting features, such as file uploading, from their networks. Developers should check with the local security staff to determine what browser scripting features are not permitted for use on the network.

1.3 DOCUMENT ORGANIZATION

This document is organized into five Sections and four Appendices. Section 1 is the introduction. Section 2 briefly describes the browsers' scripting security model, and how scripts can test or circumvent the default security of script-enabled Web browsers. Section 3 provides detailed certification policies to be used by script developers and inspectors. Section 4 provides certification procedures for the GCCS script inspector, and the integrators who load certified Web pages containing script code onto GCCS servers. Section 5 contains detailed checklists to assist developers and inspectors in verifying and certifying web pages containing browser scripts for use on the GCCS. Appendix A discusses the principles of sound software security, upon which all browser script-specific policies are based. Appendix B contains a detailed explanation of the difference between JavaScript and the Java programming languages. Appendix C contains a list of acronyms and definitions of technical terms used in this document. Appendix D contains additional references for browser script security.

2 UNDERSTANDING SCRIPT SECURITY

This section provides guidance for software proponents and systems integrators on the browser script security model, and how it is used to protect users' systems from malicious script code. It outlines current and future security features that are available with the Netscape Navigator and Microsoft Internet Explorer web browsers.

2.1 BROWSER SCRIPT SECURITY MODEL

Due to the widespread use of browser scripting, considerable effort has gone into ensuring that browser script languages are safe for use on computers worldwide. Like all complex systems, though, this has resulted in a "cat and mouse" game between web browser developers patching holes on one hand, and scientists and hackers bent on finding flaws on the other hand. Slowly, the developers are winning, as flaws in browser security have been found and patched. Due to the urgency of this race, as well as the differences between the Microsoft and Netscape browsers, it is important that system administrators stay informed on the latest developments in the industry and keep their users updated with the latest versions of software incorporating the newest security features.

2.1.1 The Browser Scripting "Sandbox"

Browser security is actually quite good from an engineering perspective. Confusion arises because the scripting languages have a large number of features that can be accessed in a large number of ways both directly through function calls and indirectly through plug-ins and ActiveX controls. These capabilities include reading and writing files on the user's hard drive, reconfiguring the browser, and running fully functional applications without the user's permission. This inherent capability of the language is limited by the browser security manager's "Sandbox," which is responsible for limiting scripts to only "safe" activities. The user's web browser implements this security policy by determining whether a given browser script is *trusted* or *untrusted. Trusted* script code has access to all the features of the scripting language, including potentially dangerous features such as reading and writing files from the user's hard drive. On the other hand, *untrusted* script code is restricted to a subset of the language with the following restrictions:

- There is no ability to read and write files.
- There is no access to file system information on the user's hard drive.
- Scripts cannot execute programs or systems commands on the user's system.
- Scripts cannot initiate network connections to computers other than the one from which the web page was loaded. (In other words, no *new* web connections are created that did not exist previously.)
- Scripts cannot access the contents of web browser windows that were not spawned from the web site that the browser is attempting access.
- Scripts cannot change all of the available web browser settings.

By restricting pages to this subset of the browser's scripting features, they are prevented from performing operations that could deny service, mislead the user, or cause compromise or

corruption of data on the user's system. Much of the debate over browser security has centered on how the available subset of features can be abused to cause a security violation. Each time a new violation is found, the available subset of the language is further restricted to make that particular security violation impossible. Consequently, the subset of the scripting languages and commands available to *untrusted* code in Netscape Navigator 4.0 and Microsoft Internet Explorer 4.0 grows increasingly smaller as more violations are discovered.

2.1.2 Degrees of Trust in Navigator and Internet Explorer

Both Netscape Navigator 4.0 and Microsoft Internet Explorer 4.0 provide the user with varying degrees of trust that can be assigned to particular web pages. In Netscape Navigator 4.0, these are referred to as *Privileges*, and a web page can ask for either a specific *Privilege*, such as the permission to write a file to the user's hard drive, or groups of privileges, referred to as a *Target*. A privilege *Target* contains a group of individual privileges; for example, the Netscape *Target* UniversalBrowserAccess includes the two *Privileges* UniversalBrowserRead and UnversalBrowserWrite. The key to this security is that the web site that is asking for such privileges must be digitally signed by the author, and that the user must explicitly grant the web site permission to use the *Privileges* or privilege *Targets* that it is requesting. This is performed through the *Privilege Request Dialog Box.* Use of these features can be risky, since untrained users can be easily confused by the complexity of this process, and coerced by a malicious web site into granting it inappropriate privileges.

In Microsoft Internet Explorer 4.0, trust of web pages is ascertained in a similar but incompatible way. Degrees of trust for particular actions are referred to as *URL Policies*, which directly parallel the Netscape *Privileges.* These policies can be manually set by the user through the View : Internet Options : Security : Custom Settings… dialog box to a high degree of granularity. Policies are grouped into four *Security Zones*: Local Computer, Local Intranet, Trusted Web Sites, Internet, and Untrusted Sites. Each *Security Zone* contains different settings for each of the *URL Policies*, corresponding to the level of trust that should be afforded to web sites within that particular zone. For example, while it is permissible for web sites within the Trusted Web Sites security zone to be able to download files to the user's system, it should not be permissible for sites within the Untrusted Sites zone to have this capability, since they should not be afforded that kind of trust. In Internet Explorer, it is possible for the user to use these settings to completely reconfigure the browser, turning off all browser-based protection of the user's computer, making it possible for JavaScript, applets, ActiveX, and browser plug-ins to have full control of the user's machine.

2.1.3 Script Security Policies

There are four security policy implementations that have been fielded with the different versions of Internet Explorer and Netscape Navigator. These policy implementations utilize very different techniques for determining the amount of trust that they give to the scripts on a given web page, and how web pages that need additional capabilities can request them, if at all:

- The *Same Origin* policy (Netscape 2.0 and up; Microsoft 3.0 and up).
- The *Tainted Code* policy (Netscape 3.0 Only).

- The *Signed Script* policy (Netscape 4.0 Only).
- The *Security Zones* policy (Microsoft 4.0 Only).

Same Origin Policy

The *Same Origin* policy dates back to the JavaScript 1.0 language that was implemented in Netscape Navigator 2.0 and Microsoft Internet Explorer 3.0. In this policy, the security of a script is determined based on the web site from which it was loaded. Scripts that are loaded from the local machine are considered to be *trusted*, while all scripts that are not loaded from the local machine are considered to be *untrusted*. Moreover, *untrusted* scripts from remote sites are only permitted to access information in other browser windows that came from the same site. For example, scripts from developer.netscape.com are not permitted to access scripts from microsoft.com, but they can be permitted to access scripts from netscape.com, since they are both part of the same site. This policy is implemented by both Netscape and Microsoft for all scripting languages, and is the only one that can be consistently counted on by web developers who are creating Internet-wide applications that may have to run on multiple platforms.

Tainted Code Policy

The *Tainted Code* policy was created in Netscape Navigator 3.0 to get around some of the limitations of the *Same Origin* policy. Specifically, the *Tainted Code* policy permits web developers to get around the *Same Origin* policy restriction forbidding web pages from different sites to intercommunicate. It does this by adding JavaScript tags to "taint" web pages and JavaScript code blocks to mark them as being safe to communicate with another site that has been similarly "tainted". In this way, a script on a web page from netscape.com can be tainted to allow communication with a script on a web page from microsoft.com. This only works, of course, if the script from microsoft.com is similarly "tainted" using special HTML tags to indicate that it is safe for scripts from netscape.com to communicate with it. The *Tainted Code* policy was only implemented in Netscape 3.0, is not available for Microsoft Internet Explorer, and has been superceded by Netscape's *Signed Script* policy and Microsoft's *Security Zones* policy.

Signed Script Policy

The *Signed Script* policy is new with Netscape Navigator 4.0, and is not available under Microsoft's Internet Explorer 4.0. With this policy, the security of a JavaScript script is determined based upon a digital signature that is applied to the script. The user determines whether a web page is to be *trusted* or *untrusted* by approving the digital signature that has been applied to the script. If the digital signature is not approved, then the web page and the JavaScript on it are considered to be *untrusted*, and operate within the security restrictions outlined above. If the digital signature is approved, then the web page and the JavaScript on it are considered to be *trusted*. However, before *trusted* web pages can utilize potentially dangerous JavaScript features, they must request explicit permission from the user, who must click in a dialog box to grant the web page security privileges to perform the potentially unsafe action. This is very different from the *Same Origin* policy in that it makes it possible for digitally signed JavaScript code that is downloaded from the web to gain the same security

privileges that were previously only available to JavaScript on the local computer. This is useful from a developer's perspective, as it makes it technically possible to write powerful JavaScript code that accesses the user's computer. On the other hand, the actual implementation is complex, and requires a technology-savvy user to differentiate web pages that legitimately require additional privileges from those that are trying to compromise security. Additionally, Navigator 4.0 incorporates a development feature called "Codebase Principals" that permits the *Signed Script* digital signature checking to be bypassed, permitting unsigned scripts to request security privileges from the user without having their digital signature checked. This feature can be abused to create a large security hole in a user's browser.

Security Zones Policy

Microsoft's second-generation security mechanism is called the *Security Zones* policy. This policy is not available using Netscape Navigator. Using the *Security Zones* policy, users of Internet Explorer 4.0 can specify groups of security privileges, called *Security Zones*, that grant web sites varying degrees of capabilities within the user's browser. Web sites from the "Local Zone" are considered to be fully *trusted*, and are granted full access to the user's web browser, while web sites from the "Untrusted sites Zone" are considered to be *untrusted*, and are granted only restricted access. This is comparable to Netscape's handling of *Signed* versus *Unsigned* Scripts, except that the user determines the zone of a particular web site, and the privileges that are to be afforded to that zone. Moreover, whereas Netscape's implementation always pops up a dialog box when a web page requests security privileges, Microsoft's implementation makes it possible for the user to disable notification of security-related activity on the part of web pages. Consequently, a user can circumvent the browser's security by reconfiguring their *Security Zones* to grant full browser access to web pages downloaded from all sites, including the Internet.

Browser Implementation Issues

Unfortunately, whereas Netscape and Microsoft have both implemented the *Same Origin* policy in compatible and consistent fashions, the *Tainted Code* policy has become obsolete, and both companies' implementations of the other two policies are completely incompatible. Consequently, web sites that have been digitally signed for Netscape Navigator 4.0 cannot be recognized and granted additional security privileges by Microsoft Internet Explorer 4.0. On the other hand, Microsoft's *Security Zones* policy can be applied to give additional privileges to sites that could not receive such privileges under Netscape Navigator. The bottom line for programmers and system administrators is that these advanced security features should be carefully used, and only in a homogeneous environment incorporating browsers from only one vendor. Future versions of the ECMAScript standard promise to bring more uniformity to the security policy implementations that are available from both Netscape and Microsoft.

2.2 DESCRIPTIONS OF POSSIBLE SCRIPT ATTACKS

Browser scripts that run within the *untrusted* browser sandbox have only limited capabilities as per the security policy of the user's browser. Scripts that run in a fully-trusted mode, on the other hand, have access to all of the features of the web browser and the user's machine. This can happen when a script requests and receives security *privileges* under Netscape Navigator, or runs in a Microsoft Internet Explorer *Security Zone* that has very few security restrictions. Additionally, scripts that download additional components such as Java applets, ActiveX controls, and browser plug-ins, can sometimes use the capabilities afforded those components to bypass the browser's security features altogether.

Consequently, there are two approaches that can be used by browser scripts to attack a user's computer. The first type of attack is that which takes place within the context of the browser "sandbox." Flaws have been found in the Netscape and Microsoft web browsers that permitted script code to access inappropriate information. These flaws in security include reading the user's Uniform Resource Locator (URL) history information, compromising the web browser's "cookie" mechanism, and manipulating browser configurations. The second type of attack takes place outside the context of the browser "sandbox." These attacks involve manipulating tools that the browser utilizes to handle certain data types or provide extended functionality to the user. As these tools are still emerging technologies, their security models are neither complete nor comprehensive. Such tools include ActiveX controls, browser plug-ins and helper applications.

2.2.1 Reading URL Histories

An early vulnerability of Netscape's original JavaScript implementation was that a script could go into the history file of the browser and identify the URLs of the web sites that the user had visited. This information could then be used by the script's author, or anyone else, without the user's knowledge or consent. This security hole was fixed with the Netscape 2.02 browser, and is no longer exploitable by web developers.

2.2.2 Accessing A User's Cookie

An Internet 'cookie' is a *name=value* pair that is kept in a file which is maintained by the browser. This information is tied to a particular URL and is returned automatically to the web server whenever the user visits that site. In early versions of the Netscape web browser, it was possible for a JavaScript programmer to "steal" an Internet cookie that was placed in the user's browser cookie file by another web site. Using this technique, a cookie thief could then impersonate the original user, possibly gaining access to sensitive information on other web sites that was previously only accessible to the original user. This security hole was fixed with the Netscape 2.02 browser, and is no longer a vulnerability.

2.2.3 Alteration of Browser Settings

Scripts can be used to manipulate the browser and its user interface, including opening new windows, closing existing windows, and turning on or off the visual elements of the browser screen. At a minimum, these capabilities can be irritating, resulting in windows that are inconveniently placed, or confusing user interfaces. However, at their worst, these capabilities can be used to damage the system itself, or trick a user into doing damage. Damaging techniques include creating arbitrarily small windows that permit a web page to stay active unbeknownst to the user, creating windows that look like other application windows, or password entry windows, or manipulating the browser's main window's scroll bars, menu bars, and status bars to such a degree that the user cannot properly control the browser program. The security policy enforcement of modern browsers make these capabilities impossible to execute. However, if the security policy enforcement of the browser is deactivated, scripting languages have the capability to perform all of these actions.

2.2.4 Script Interaction with ActiveX Controls

Microsoft's ActiveX technology is a way for creating portable, reusable, scriptable cross-platform controls. It can be used to implement functionality in a variety of environments, including the Windows operating system, Microsoft applications such as Word and Access, and also the Internet Explorer Web Browser. ActiveX can also be used within Netscape Navigator via a browser Plug-In. The problem with ActiveX controls is that they are written in low-level languages, and interface directly with the computer hardware and operating system. Because they communicate directly with the operating system, their only interaction with the web browser's security mechanism is when they are downloaded. At that time, the user has the option of accepting or denying the ActiveX control. However, once it has been accepted by the user, a single malicious ActiveX control is all it takes to wreak havoc on a user's system. Web developers can use ActiveX controls to perform illicit actions in two ways: first, they can design malicious ActiveX controls and hope that the user will accept one, and second, they can use browser scripting to make normal ActiveX controls perform malicious actions.

2.2.5 Script Interaction with Plug-ins and Helper Applications

Plug-ins are similar to ActiveX controls, but are specific to web browser programs. Plug-ins permit scripts and HTML tags to interact with other programs, or mini-programs, installed on the user's computer. For example, when a web page loads a sound file, a sound player plug-in, such as Netscape's LiveAudio, is used to play the sound file to the user. Similarly, when a user clicks on an Adobe Acrobat file, the Adobe Acrobat plug-in is used to show the contents of that file within a web browser window. Plug-ins, like ActiveX controls, can be manipulated and controlled via browser scripting. Consequently, some plug-ins, such as *Carbon Copy/Net*, can be used to compromise the security of the user's computer. Users must be careful and cognizant of what plug-ins they use on their systems.

3 SCRIPT CERTIFICATION POLICIES

This section provides guidance for developers and code inspectors to mitigate the security risks posed by browser scripting software. It explains the policies that should be followed when developing browser scripts that will operate on GCCS machines.

3.1 GENERAL SCRIPT SOFTWARE DEVELOPMENT GUIDANCE

Browser scripts must adhere to the same development guidelines as code written in other languages for use within the GCCS. The "GCCS Security Guidelines for Developers" document provides details on security requirements for code development for programs to be deployed in the GCCS. IEEE 12207, *U.S. Software Lifecycle Process Standards*, which replaced MIL-STD 498, *Software Development and Documentation*, in May, 1998 provides general guidance on the implementation of software systems for military use. These documents provide recommendations to reduce the risks associated with implementing distributed systems, and highlight the risks involved with using software that may have been obtained from untrusted sources. An example of untrusted software is a browser script library that was downloaded from a non-commercial Internet site and then used within a larger project without a proper security review of the script code contained within the library.

3.2 SCRIPT CERTIFICATION PROCESS

The "GCCS Security Guidelines for Developers" document identifies two types of certification:

- Derived certification
- Inspection certification

All mobile code developed in browser scripting languages is subject to certification by one of these processes prior to installation on a GCCS server. Code requiring inspection is subjected to code analysis by the Defense Information Systems Agency (DISA) D6 or an appointed representative. All testing must be conducted on isolated systems that are not connected to the operational GCCS. Some tests may require an independent expert in JavaScript, JScript or VBScript to inspect the code. If an independent expert is not available, the software proponent should consult with the software developer to ensure that all browser scripts are in compliance with the policies described in this section. If unsatisfactory results are obtained in this inspection process, the software is returned to the developer for corrective action.

Upon acceptance of the code analysis by DISA D6, browser scripts are digitally signed with an X.509 certificate authenticating their certification for use on the GCCS. The digital signature can then be validated at any time to verify the authenticity of the browser script software.

3.3 SCRIPT CERTIFICATION POLICIES

The following are minimum guidance for browser scripting use within the GCCS. These policies must be adhered to by web developers, or justification must be provided documenting the reasons that they cannot be followed.

#1: Script Testing

Policy: Browser scripts must be tested in a stand-alone environment to ensure that no damage is done to essential systems. Scripts must not crash or disrupt the use of either Netscape or Microsoft web browsers. This includes the triggering of run-time error windows that require user intervention to continue.

Rationale: The use of untested or undocumented scripts dramatically increase the risk of a potential catastrophic failure or data loss. Examining script code in a test environment reduces the risk of loss or corruption to a single machine. Correctly written browser scripts should not disrupt or crash the user's web browser; scripts that cause such crashes or disruption cannot be certified for use on GCCS. Script code that causes run-time errors may lead to increased risk by confusing users or influencing them to change their browser's security settings inappropriately in an effort to resolve the problem.

#2: Certified Script Libraries

Policy: Browser scripts embedded in web pages may call script "library code" that is separated from the web pages and contained in separate script files. These script libraries must be certified to the same policies as the web pages that call them. Libraries that were previously certified and are re-used in a new project may be re-used without recertification provided that the original library source code has not been changed, and any previously applied certification digital signatures that were applied to the library file are still valid.

Rationale: It is possible to create browser script libraries that can be certified and reused. It is in the interest of developers to re-use such code in new software development projects, as this reduces the amount of new software code that must be written. It is in the interest of DISA D6 to encourage such practice, as it simplifies the process of certifying such products for use on the GCCS.

#3: Using Browser Script to Access Java Applets, Objects and Methods

Policy: Java code – including applets, standalone objects, and the methods of those objects – that is called from within browser scripting code must meet the specifications established by GCCS in the *Mobile Code Security Policy Guidance for Java Software Development*. Java objects and methods that are built into the web browsers and the operating systems of users' machines may be accessed by browser script code provided that that the operation of those objects is in accordance with the Java policy guidance, and their employment from the browser script is in accordance with the remainder of this document's policy guidance.

<u>Rationale:</u> Browser script code can interface with Java code, including applets, standalone objects, and operating system objects. This is accomplished using the built-in features of Microsoft Internet Explorer and the LiveConnect plug-in for Netscape Navigator. These actions are constrained to the security model of the Java "Sandbox" of the user's browser, which may not be set to the same security restrictions as the rest of the browser. Specifically, if Java's sandbox security is disabled by the user, this technique can be used to bypass the web browser's script security, permitting:

- Scripts to interact with the standard Java system classes built into the browser.
- Scripts to interact with Java applets by reading and writing public fields of the applet and invoking public methods.
- Scripts to interact with other Java-enabled browser plug-ins in the same way.
- Applets and Java-enabled plug-ins to interact with browser scripts by reading and writing script object properties, array elements, and invoking functions.

#4: Using Browser Scripts to Access ActiveX Controls

<u>Policy:</u> Browser scripts may access ActiveX controls provided that the controls themselves are in compliance with the GCCS ActiveX policy, and the controls are used in a manner that is consistent with the remainder of this policy. Scripts may not attempt to download ActiveX controls that have not been digitally signed by their authors and approved by for use on the GCCS. The user must have the opportunity to approve or deny all downloads of ActiveX controls to their machines, and scripts must degrade gracefully when the desired ActiveX controls are not permitted by the user.

<u>Rationale:</u> ActiveX controls are not subject to the browser's "sandbox" security models that are applied to browser scripts or Java applets that run on users' machines. Consequently, a malicious ActiveX control could do considerably more damage than malicious controls in other languages. Similarly, benign ActiveX controls that have the capability to compromise or corrupt users' data may be misused by malicious browser scripts or Java code that control them. Both of these cases must be prevented. Finally, users must have control over the downloading and installing of such ActiveX controls, such that the users' denial of a particular ActiveX control must not cause browser scripts to fail catastrophically.

#5 Using Browser Scripts to Access Browser Plug-Ins

<u>Policy:</u> Browser scripts may access browser plug-ins provided that the plug-ins are used in a manner that is in compliance with the remainder of this policy. Browser scripts may not attempt to download and install browser plug-ins that have not been approved for use on GCCS computers. DISA D6 must approve all downloads of browser plug-ins to the GCCS machines. Browser scripts must fail gracefully if the user does not permit a required plug-in to be installed.

<u>Rationale:</u> Browser plug-ins are not subject to the browser's "sandbox" security models that are applied to browser scripts or Java applets that run on users' machines. Consequently, a

malicious plug-in can do considerably more damage than malicious browser scripts or Java applets. Similarly, benign plug-ins that have the capability to compromise or corrupt users' data may be misused by malicious scripts or Java applets that control them. Both of these cases must be prevented. Finally, the user must have the opportunity to grant or deny permission before any new plug-ins are added to their browser.

#6: Browser Information Access

Policy: The use of browser scripts to access or write the browser program's internal information is prohibited. This includes access to the browser's "history file" or data used when filling out on-line forms. Also included in this category are Browser Scripts which can be used to move or resize a browser window outside the viewable area or disguise browser windows to appear similar to those used by other programs.

Rationale: Private information may be obtained through the use of browser scripts that access the browser's "history" or information entered into on-line forms by users. Browser scripts that modify the appearance of a browser window in an attempt to make them impossible to understand or appear to be windows from programs, can confuse users. A denial of service attack could consist of relocating or resizing browser windows to make them unavailable to the user.

#7: Access to the User's Local Files

Policy: Browser scripts which attempt to read, relocate, or write files to or from the browser's host machine or devices connected to it are prohibited. This does not include the use of browser cookies.

Rationale: Scripts which attempt to read, relocate, or write files to or from the browser's host machine or devices connected to it are sources for potential data compromise, loss, or corruption. This is a large risk when operating in a protected environment. The browser's cookie mechanism provides additional security restrictions that mitigate such risk, and consequently its use is acceptable.

#8: Browser Preference Access

Policy: Browser scripts must not attempt to access, modify, or write to the user's browser settings or preferences without a legitimate need and the user's consent.

Rationale: Information stored in a browser's preferences may be used to gain information about a specific user or the system(s) on which they work. Such information may include a user's name, electronic mail (e-mail) address and server connectivity, as well as security level settings. Once harvested, this information can be employed in attempts to gain or deny access in other ways. Additionally, reconfiguration of the browser's settings may be used to either confuse the user or deny them use of their web browser.

#9: Access to Browser Privileges and Special Security Features

<u>Policy:</u> Browser scripts must not require access to customized security privileges or other browser security features without written justification from the developer. In the event that the use of such features is required, the browser scripts must be written such that the features are enabled just prior to use, and then immediately disabled immediately after use. When these features are enabled, the user must be notified on-screen, and given the opportunity to deny them, in which case the browser script must fail gracefully.

<u>Rationale:</u> Most browser scripts operating on GCCS should not require access to advanced browser security features, and any such use must be examined to determine if it is really necessary. In the event that such use is required, minimizing the length of availability and amount of access to the more risky browser privileges will minimize the total risk that is involved.

#10: Automatic E-mail

<u>Policy:</u> Browser scripts must not use the browser's e-mail sending capability to send e-mails that contain hidden fields or surreptitious information from the user's e-mail account. Use of this capability must be with the user's permission, such that the user must have the opportunity to view the complete text of the e-mail that is to be transmitted and have the opportunity to change the text of the message before transmission, or cancel the transaction altogether.

<u>Rationale:</u> Browser scripts, which automatically send or populate e-mail fields, increase the risk of potential data compromise. Such scripts may allow e-mail to be automatically sent from a user's account, in their name, on their behalf. Automatic population of form fields with user information as well as the automatic sending of e-mail provides a simple method for transporting private information off of an unsuspecting user's machine.

4 SCRIPT CERTIFICATION PROCEDURES

This section provides specific procedures to be followed by browser script inspectors to certify scripts as safe for use on GCCS. These procedures ensure that the subject code meets the policies specified in Section 3 above, and can be followed roughly in sequence.

4.1 OBTAIN SCRIPT CODE FROM VENDOR OR WEB PAGE

Obtain the complete source code for the software project, including all web pages and browser scripts. Request any available documentation which pertains to the code including any information regarding the use of pre-certified code libraries. In order to properly inspect browser script code, it is necessary to examine both the scripts themselves and the web pages that call them, as the manner in which a script is employed may determine the level of risk that it poses to the user.

4.2 INSPECT THE BROWSER SCRIPT CODE

Inspect the browser scripts according to the policies in this document. Following the procedures below, use the Inspection Checklist provided in Section 5, and the software certification policies in Section 3 as required. It may be necessary to consult the web developer during this process.

4.2.1 Test the Script Code

(Ref: Policy #1)

Install the current GCCS-approved Web Browser (Netscape Communicator 4.x) onto a non-essential network computer that is isolated from the GCCS operational network. Use the web browser to view the web pages in question, running the browser scripts that are accessed by them. Become familiar with the purpose and functionality of the web pages that are being tested. Observe any abnormal browser actions including, crashing, “hanging,” or the triggering of multiple run-time error windows requiring user intervention. This test may be impossible if the browser scripts are provided “as-is,” and are not attached to a web page. If this is the case, it must be noted.

4.2.2 Browser Script Reuse

(Ref: Policy #2)

Examine the browser script source code for calls to other previously written browser scripts or Java applets. First, determine if each called script is a member of the GCCS-approved and certified library. If it has already been incorporated into the library, ensure that the code has not been modified by validating its GCCS-assigned digital signature. All browser scripts must be inspected or re-inspected for compliance with the policies in Section 3 if the digital signature is no longer valid or if the script has never been tested and signed.

4.2.3 Use of Java Applets, Objects, and Methods

(Ref: Policy #3)

Check scripts for the use of Java code including applets, standalone code, as well as objects or their methods. If any calls or references to Java code are found, ensure that the items referenced are compliant with the specifications in the *Mobile Code Security Policy Guidance for Java Software Development* and are used in accordance with this document.

4.2.4 Use of ActiveX Controls

(Ref: Policy #4)

Inspect browser scripts for the use of ActiveX controls. All ActiveX controls that are accessed by the browser script must conform to the GCCS ActiveX policy. Browser scripts should not try to download ActiveX controls without the user's consent, and should degrade gracefully if the user does not grant permission for downloading. Use of those ActiveX controls must be consistent with the remainder of this policy to ensure that potentially dangerous ActiveX controls are not abused by the scripts that call them.

4.2.5 Use of Browser Plug-Ins

(Ref: Policy #5)

Inspect browser scripts for the use of browser plug-ins. Only plug-ins that have been approved for use on GCCS may be accessed from browser scripts. Browser scripts should not try to download plug-ins without the user's consent, and should degrade gracefully if the user does not grant permission for downloading. Use of those plug-ins must be consistent with the remainder of this policy to ensure that potentially dangerous capabilities are not abused.

4.2.6 Browser Information Access

(Ref: Policy #6)

Examine browser scripts for code which attempts to read or modify the browser's internal information stores. These include the browser's "history" file, information stored from the completion and use of on-line forms, and the look, appearance and size of browser windows. Check that these capabilities are not abused to obtain inappropriate information, confuse the user, or cause a denial of service attack.

4.2.7 File Read and Write

(Ref: Policy #7)

Check the scripts to ensure that files are not read or written on the user's system without the user's knowledge and consent. The use of "cookies" is allowed, but the implementation and intended use must be examined for potential security violations.

4.2.8 Browser Preference Access

(Ref: Policy #8)

All browser scripts must be examined for code which attempts to access, modify, or write to a browser's settings or preferences. If any access is granted, the information being requested

or written should be examined for private or security related content. The implications of the increased privileges must be determined and accepted before a script can be certified and signed.

4.2.9 Browser Privileges and Security Features

(Ref: Policy #9)

Check that browser scripts requesting access to browser access privileges and other special security features request a user's permission before altering them. If permission is granted, special accesses and permissions must be enabled just prior to use and then immediately disabled after use. The need for such browser privileges must be justified and documented by the developer.

4.2.10 Automatic E-mail

(Ref: Policy #10)

Check that browser scripts ask the user's permission before automatically sending e-mail. If e-mail fields are populated on the user's behalf, the user must have the opportunity to edit these fields before transmission, and also to cancel the transmission of the message after viewing it. The use of such automatic e-mail features must be justified and documented by the developer.

4.3 DIGITALLY SIGN APPROVED BROWSER SCRIPTS

Browser scripts that are approved according to section 4.2 above can then be digitally signed by the inspecting authority using a GCCS-approved digital certificate. This digital signature is affixed to the code to show that it has passed the certification process.

4.4 LOAD CERTIFIED BROWSER SCRIPTS ON THE GCCS

DISA D6 maintains a Web page that contains all Scripts that have been tested and certified for use within the GCCS. Load all certified and signed browser scripts onto the GCCS site in order to make it available as a resource for use by other GCCS web developers.

5 CODE CHECKLIST FOR DEVELOPERS/INSPECTORS

This checklist is provided to assist developers and inspectors in ensuring that browser scripts are used in accordance with the policies of this document.

Table 1. Browser Script Inspection and Certification Explanations

Test	Actions	Expected Results	Action if Results Not as Expected
1 Test code. (Ref: Policy #1)	Test web pages containing browser scripts on GCCS-approved web browser.	No run-time errors, security violations, or system crashes encountered. Web pages do not appear to compromise browser security.	Return web pages to developer for modification.
2 Check library references. (Ref: Policy #2)	Ensure that all browser script libraries are already certified for use on GCCS, and digitally signed as necessary.	All libraries are certified and signed.	Certify the libraries according to the *GCCS Mobile Code Security Policy Guidance.*
3 Check library use. (Ref: Policy #2)	Inspect the use of such libraries, ensuring that library calls that pose risk to the user are employed in a safe manner.	All library calls are safe and pose no risk to the user.	Return web pages to developer for modification.
4 Use of Java programs, applets, objects, or methods. (Ref: Policy #3)	Examine code for the use of Java programs, applets, objects, or methods.	No Java references found.	Ensure that Java use is in compliance with the *Mobile Code Security Policy Guidance for Java Software*.
5 Use of ActiveX controls. (Ref: Policy #4)	Examine code for the downloading or use of ActiveX controls.	Only standard GCCS-approved ActiveX controls are used.	Ensure that non-standard ActiveX controls are inspected for compliance with the *Mobile Code Security Policy Guidance for ActiveX Controls*.
6 Use of browser plug-ins. (Ref: Policy #5)	Examine browser scripts for the downloading or use of browser plug-ins.	Only standard GCCS-approved plug-ins are used.	Ensure that non-standard browser plug-ins are certified for use on GCCS.
7 Browser "History" access. (Ref: Policy #6)	Examine browser script for code that attempts to access inappropriate information from the browser's "History" files, or on-line forms that do not belong to the parent web site.	Script does not make inappropriate use of the "History" file or on-line forms information.	Return web pages to developer for modification.
8 Browser configuration changes. (Ref: Policy #6)	Examine scripts for code which attempts to modify the location, size, or appearance of browser windows.	The browser's main window is not manipulated; sub-windows are not modified in such a way as to mislead the user.	Return web pages to developer for modification.

Test	Actions	Expected Results	Action if Results Not as Expected
9 Use of "Cookies". (Ref: Policy #7)	Check for use of "cookies" to access information that does not belong to the parent web site.	"Cookies" are used safely.	Return web pages to developer for modification.
10 File read or write. (Ref: Policy #7)	Examine browser scripts for code used to move, relocate, or write files to or from the host machine or connected systems.	Files on the user's machine are not manipulated inappropriately.	All file-access requirements must be documented. If they are inappropriate, return web pages to developer for modification.
11 Browser preference access. (Ref: Policy #8)	Examine scripts for code that attempts to access, modify, or write to a browser's settings or preferences.	Browser settings are not changed inappropriately.	All reconfiguration of the browser's settings must be documented. If they are inappropriate, return web pages to developer for modification.
12 Use of browser security privilege features. (Ref: Policy #9)	Examine scripts for code that attempts to use browser security privilege features.	Security privileges are not used inappropriately.	All use of security privileges must be documented. If the use is inappropriate, return web pages to developer for modification.
13 Minimize use of security privileges. (Ref: Policy #9)	Check that security privilege features are enabled just prior to use and disabled immediately after use.	Security privileges are enabled just prior to use and disabled immediately after use.	Return web pages to developer for modification.
14 Use of automatic e-mail. (Ref: Policy: #10)	Examine scripts for code that automatically sends e-mail, or populates the fields of an e-mail that is to be sent by the user.	Scripts should not send e-mail or populate e-mail fields without the user's knowledge and permission.	Return web pages to developer for modification.

Table 2.   Browser Script Inspection and Certification Checklist

Browser Script Package	
Developer	
Developer Contact Info	Address: Phone / Fax: E-Mail / Web:
DISA/JS Proponent	
Inspection Date & SW Version	
Inspector / Certifier	
Approval Date	
Digital Certificate Used	

	Test	Passed	Failed	Comments
1	Test code. (Ref: Policy #1)			
2	Check library references. (Ref: Policy #2)			
3	Check library use. (Ref: Policy #2)			
4	Use of Java programs, applets, objects, or methods. (Ref: Policy #3)			
5	Use of ActiveX controls. (Ref: Policy #4)			
6	Use of browser plug-ins. (Ref: Policy #5)			
7	Browser “History” access. (Ref: Policy #6)			
8	Browser configuration changes. (Ref: Policy #6)			
9	Use of “Cookies”. (Ref: Policy #7)			
10	File read or write. (Ref: Policy #7)			
11	Browser preferences access. (Ref: Policy #8)			
12	Use of browser security privilege features. (Ref: Policy #9)			
13	Minimize use of security privileges. (Ref: Std #9)			
14	Use of automatic e-mail. (Ref: Policy: #10)			

APPENDIX A
SOFTWARE SECURITY PRINCIPLES

This section describes the security principles upon which specific software certification policies are derived. These principles are language independent, and can apply to software of all types. They are derived from ISO/IEC 12207 *Information Technology Software Life-cycle Processes*, which replaced MIL-STD 498, *Software Development and Documentation* in May, 1998. There are four areas of concern that must be addressed through the software certification process. They are: denial of service, unusual behavior, the compromise of sensitive data, and the corruption of sensitive data. All of these areas of concern apply to both the user's computer as well as other users' computers connected through a network or other medium.

Denial of Service

Principle: Software shall not cause software, peripherals, or the central processing unit of the user's or connected computers to become unavailable. The software developer shall document all software incompatibilities that result in denial of service to the user.

Rationale: This condition exists when the subject application software restricts the user's ability to access the programs and data stored on either their system or the network. An example of such behavior is a program that disrupts the computer's ability to print or access the network. Another example of such behavior is a program that permits the user to enter data into it, then crashes before the user can finish his or her work.

Misleading Behavior

Principle: Software shall not have unusual behaviors that are used to maliciously mislead users to perform actions that compromise or corrupt the data on either their systems or other systems attached via a network.

Rationale: This condition exists when the subject application software behaves unusually in an effort to confuse the user or persuade the user to perform actions that would endanger data stored on the network. An example of such behavior is a program that puts up a dialog box requesting the user to send data to a third party, or one that requests the user to obtain inappropriate data from other users. Using these techniques, a program that does not cause data corruption or compromise on its own can indirectly cause such events by persuading users to do them on its behalf.

Compromise of Data

Principle: Software shall not compromise the data stored on the user's or connected systems without the user's explicit permission. Such permission will identify the data that is to be sent and the reason for the transmission. This behavior shall be clearly explained in the software documentation.

Rationale: This condition exists when the subject application software causes data on the user's or connected systems to be compromised without the user's consent, and possibly without the user's knowledge. An example of such behavior is a program that copies user files to another location on the network without the user's knowledge, or a program that surreptitiously records the user's behavior in order to capture passwords and other sensitive information.

Corruption of Data

Principle: Software shall not corrupt the data stored on the user's or connected systems without the user's explicit permission. Such permission will identify the data that is to be destroyed and the reason for its destruction. This behavior shall be clearly explained in the software documentation.

Rationale: This condition exists when the subject application software causes data on the user's or connected systems to be corrupted or destroyed in such a way that it is no longer of value to the user. Examples of such behavior include reformatting of the user's hard drive or deletion of user files without the user's permission.

APPENDIX B
JAVASCRIPT VERSUS JAVA

Due to the rapid pace of development in the Internet community and the plethora of hype that accompanies that development, a large number of new buzzwords have erupted onto the public scene. These buzzwords are presented often without proper definition or differentiation, and seldom with good descriptions of how “it all comes together” into solutions that end-users can use to solve real-world problems.

Two words that have become almost hopelessly embroiled in the hype wars are JavaScript and Java. Due to the efforts of Sun and Netscape to counter Microsoft’s tremendous market clout, they have both become the foot soldiers of a cyber-war where - if one is to believe the press releases – the future of information availability is at stake. The remainder of this section attempts to describe how JavaScript and Java differ, and how they both fit into the big picture of web development.

The Basics

To begin, both JavaScript and Java are *languages* that permit computer instructions to be downloaded over the Internet, and run on a user’s local computer. Both of these languages follow a similar syntax, which means that software code that is written in JavaScript looks a lot like software code that is written in Java. However, the similarity ends there, as the languages are compared in the table below:

	JavaScript	**Java**
Language Type	Object-based	Object-oriented
Creator	Netscape	Sun Microsystems Inc.
Used for	Scripts within a browser	Standalone applications, Applets, Servlets
Interaction with Web pages	Script placed within HTML file; can manipulate HTML and can act as a liaison between applets and HTML elements.	Separate program called by HTML file; confined to specific region of browser; cannot manipulate HTML elements
Security Model	Object Signing (Netscape) Authenticode (IE)	Sandbox

Table 3 Java and JavaScript Differences

Java

Java is a general-purpose programming language that is written by a programmer, compiled into an intermediate byte-code, downloaded over the web, and run on a user’s machine by the *Java Virtual Machine*. Sun created Java with a large number of virtual machines that permit the same Java code to run on almost any platform, including Apple’s Macintosh, Sun Solaris, Unix, Linux, and of course Windows, using both the Netscape Navigator and Microsoft Internet Explorer browsers. With this large number of virtual machines that can run Java, it is a

truly portable language that permits programmers to create general-purpose software for any platform.

JavaScript

JavaScript, on the other hand, is a web browser scripting language that has the sole purpose of giving web pages additional functionality when viewed on the Netscape Navigator web browser. Microsoft's web browser did not support JavaScript until Internet Explorer 3.0 which supported *JScript*, which is a Microsoft-only dialect of the language, as opposed to a straight implementation. This scripting code is used to instruct the browser to perform additional operations, such as popping up a window, or scrolling a message, that cannot be done with straight HTML alone. Ironically, for a web page to incorporate a *Java* applet, *JavaScript* code must be written to tell the web browser to run the applet!

Differences

Both languages treat program elements – such as icons or pop-up windows – as objects, which can pass instructions to one another. But a true object-oriented language, as Java is, also makes heavy use of inheritance. Inheriting functionally from existing objects and adding new attributes can extend objects. JavaScript does not have this ability. JavaScript objects can be created, but cannot inherit any other object's properties.

The languages also differ in how they interact with browsers and web pages. Java, which requires a compiler (a program that translates human-readable code into machine-readable executables), can be used to create either stand-alone applications or applets that run with a browser. JavaScript, on the other hand, works only within a browser and is not compiled. It cannot be used to develop standalone applications. JavaScript is a scripting language for writing short programs, or scripts, such as log-on procedures.

Java applets are downloaded as separate files onto a client machine and are executed independently of HTML files and images; they are not visible in the source file. JavaScript is embedded within the HTML file and visible in a document's source. One can use JavaScript to manipulate all of the HTML elements on a Web page. A Java applet; however, is a self-contained application that lies within a web page, and is limited to a small space on the web page's window. Whereas Java applets can only handle commands that occur within the window's boundaries, JavaScript can capture events anywhere in the browser, and then even pass those events on to an embedded Java applet.

Security

While Java has a formal security model, JavaScript security is based on the implementation of the browser. Several problems have been reported with JavaScript, including the ability to upload or retrieve arbitrary files from a user's machine. These have been reportedly fixed in newer browsers, but denial of service attacks are still possible.

JavaScript differs from Java in several important ways that relate to security.

- Java signs classes and is able to protect internal methods of those classes through the public/private/protected mechanism. Marking a method as protected or private immediately protects it from an attacker. In addition, any class or method marked *final* in Java cannot be extended and is thus protected from an attacker. On the other hand, because JavaScript has no concept of private and public methods, there are no internal methods that could be protected by simply signing a class. In addition, all methods can be changed at runtime, so they must be protected at runtime.

- In JavaScript one can add new properties to existing objects, or replace existing properties (including methods) at runtime. This is unavailable in Java. Once again, protection that is automatic in Java must be handled separately in JavaScript.

Conclusions

Despite all the hype, both JavaScript and Java are important parts of the present and future of the Internet. They are complementary tools that each have a place in web development, and programmers must use both of them to create effective web-based applications. Due to their differences in purpose, implementation, and application, network administrators must pay careful attention to how both languages are both utilized within their networks, and should stay abreast of the rapid evolution of the underlying technologies.

APPENDIX C
ACRONYMS AND DEFINITIONS

TERM	DEFINITION
CERT	Computer Emergency Response Team
CSS	Cascading Style Sheet
DISA	Defense Information Systems Agency
ECMA	European Computer Manufacture’s Association
E-Mail	Electronic Mail
GCCS	Global Command and Control System
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
IE	Internet Explorer
IP	Internet Protocol
SIPRNET	Secret Internet Protocol Router Network
SSJS	Server Side JavaScript
URL	Uniform Resource Locator
WWW	World Wide Web

APPENDIX D
SCRIPT SECURITY REFERENCES

References

Flanagan, D., D. Shafer. J*avaScript: The Definitive Guide.* Sebastopol, CA: O'Reilly and Associates, (1996).

Danny Goodman, *JavaScript Bible*, IDG Books, 3rd Edition (1998).

Web Sources

The following are useful Uniform Resource Locators (URL) for JavaScript and security related topics:

Netscape
http://developer.netscape.com/docs/manuals/communicator/jsguide4
http://www.netscape.com/products/security/index.html

JavaScript Security
http://www.osf.org/~loverso/javascript, 14 Nov 97

CERT Notifications
http://www.cert.org

The WWW security FAQ
http://www.w3.org/security/faq/www-security-faq.html

Princeton University
http://www.cs.princeton.edu/sip/

International Computer Security Association
http://www.ncsa.com

Bugtraq
http://www.geek-girl.com/bugtraq

Microsoft
http://www.microsoft.com/security